Grado en Ingeniería Informática
Grado en Matemáticas e Informática

Asignatura: PROGRAMACIÓN II

Arrays de Objetos

Profesores de Prog II

DLSIIS - E.T.S. de Ingenieros Informáticos
Universidad Politécnica de Madrid

Febrero 2015

Arrays de objetos

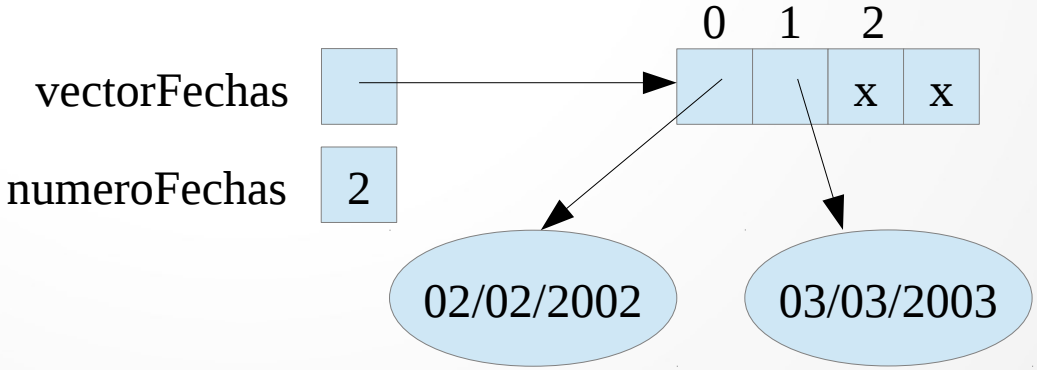
```
public static void main(String[] args) {

  Fecha [] fechas; // declaración del array

  fechas = new Fecha[3];

  fechas[0]=new Fecha(1,1,06);
  fechas[1]=new Fecha(2,1,06);
  fechas[2]=new Fecha(3,1,06);

  for (int i=0; i<fechas.length; i++)
  System.out.println(fechas[i]);
}
```

Se crea espacio para tres referencias a Fecha

Se crea la primera instancia y es referenciada por fechas[0]

Arrays parcialmente llenos

- **Una vez creado un vector no cambia de tamaño**
- Para cambiar el número de elementos se puede definir el vector con capacidad para el máximo número de elementos, pero sólo se ocupan los primeros
  - Se puede usar una variable que lleve la cuenta de los elementos que contiene el vector
  - Sólo se procesan los elementos ocupados, no se necesita recorrer el vector entero
  - Si tenemos un vector de objetos Fecha con capacidad de 4, pero solo hemos guardado 2 Fechas, debería ser:

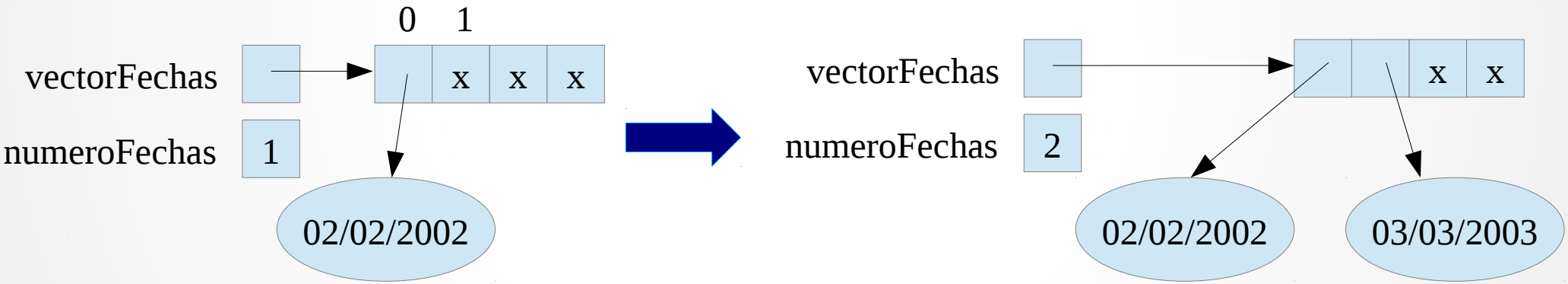


Creación del *array*

- Hay que prever el número **máximo** de elementos que pueden almacenarse en el vector
  - Se declarará un *array* vacío, con la capacidad dada
  - Inicialmente el número efectivo de elementos en el *array* será 0

Añadir un elemento

- Si se quiere **insertar** un nuevo elemento, dicho elemento debe meterse al **final** de los existentes
  - Si el *array* actualmente contiene **1** elemento y se quiere introducir otro, el nuevo se meterá en la posición ... **1**
  - Se actualiza el número de elementos en el *array*

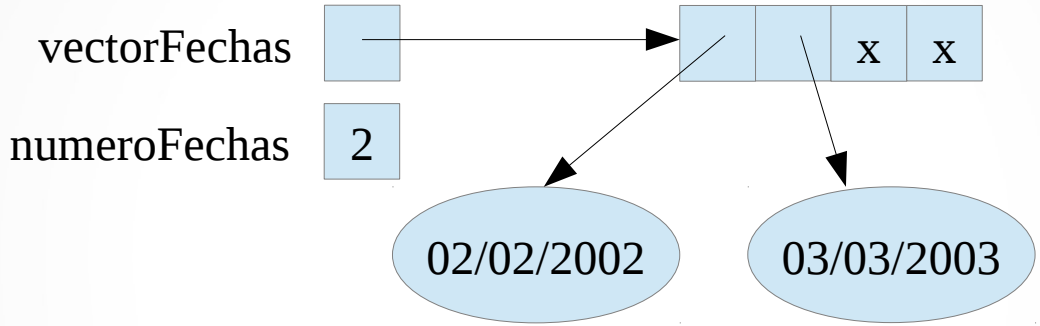


Añadir un elemento

- Si se quiere **insertar** un nuevo elemento, dicho elemento debe meterse al **final** de los existentes
  - Si el *array* actualmente contiene **n** elementos y se quiere introducir otro, el nuevo se meterá en la posición ... **n**.
  - Se actualiza el número de elementos en el *array*

Recorrido del *array*

- Si se va a **recorrer** el vector
  - Desde la posición 0 hasta la posición *numElementos*-1.
  - Ejemplo: Si tenemos el vector

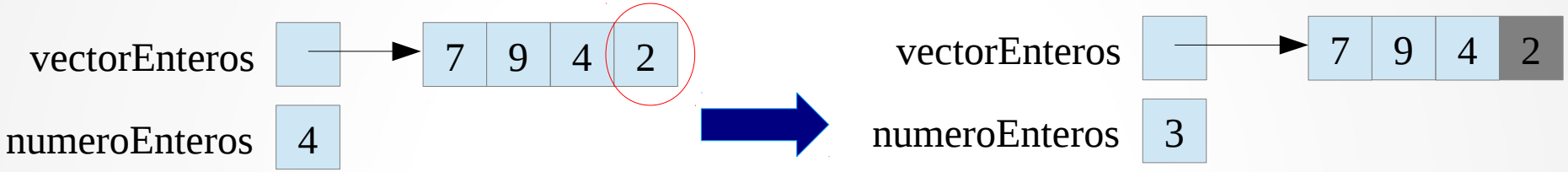


Se recorrerá desde la posición 0 hasta la 2-1 (1)

```
for (int i=0; i<numeroFechas; i++) {
    System.out.println(vectorFechas[i]);
}
```

Eliminar un elemento

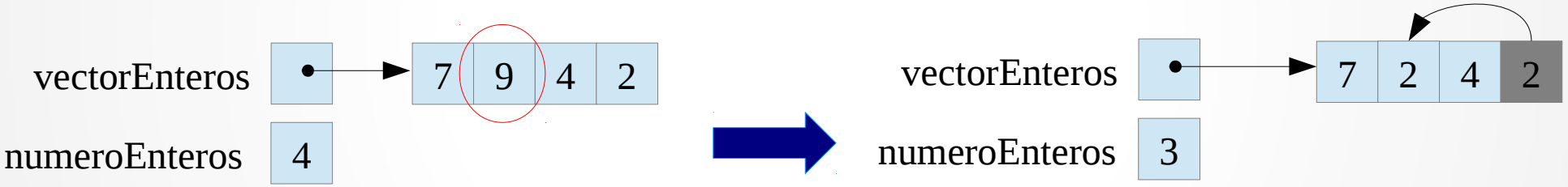
- Caso particular: se quiere **borrar** el **último** elemento del vector
  - Basta con decrementar el número de elementos



  - Cuando lo recorramos pararemos uno antes, y no se accederá al cuarto elemento

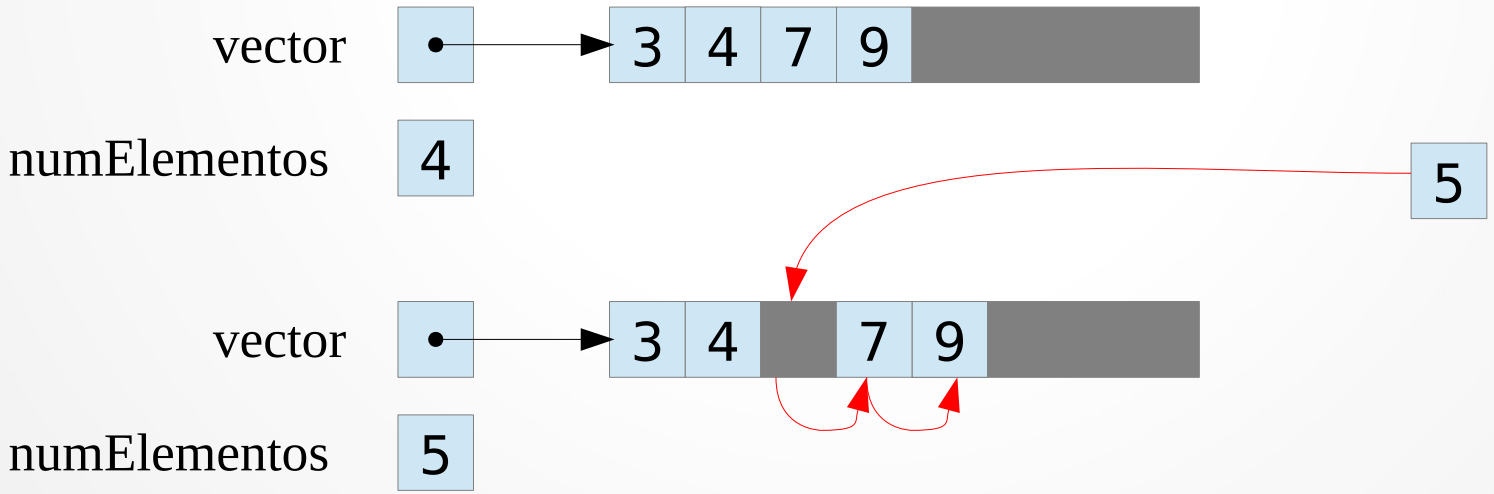
Eliminar un elemento

- Caso general: **borrar** un elemento que **no es el último**; debe rellenarse un hueco dentro del *array*:
  - Solución simple: **desplazar** el elemento situado en la **última** posición **a la posición que se quiere borrar**.
  - Se decrementa en uno el número de elementos en el *array*



Ordenación por inserción (construcción)

- Para aprovechar las ventajas de la búsqueda binaria los elementos del vector deben estar en orden.
  - Se pueden ir manteniendo en orden mientras se añaden elementos al vector.
  - Para mantener el orden hay que mover una posición hacia el final cada uno de los elementos (potencialmente objetos) mayores (con el tipo de ordenación entre objetos que se defina) que el que se inserta.



Insertar un elemento

```
int[] vector = ....;
int numElementos = ....;
int nuevo;

int pos = numElementos;
while (pos > 0 && vector[pos-1] > nuevo) {
    vector[pos] = vector[pos-1];
    pos--;
}
vector[pos] = nuevo;
numElementos++;
```

Retirar un elemento

- Hay que mover una posición hacia el principio cada elemento detrás del que se elimina, para rellenar el hueco.

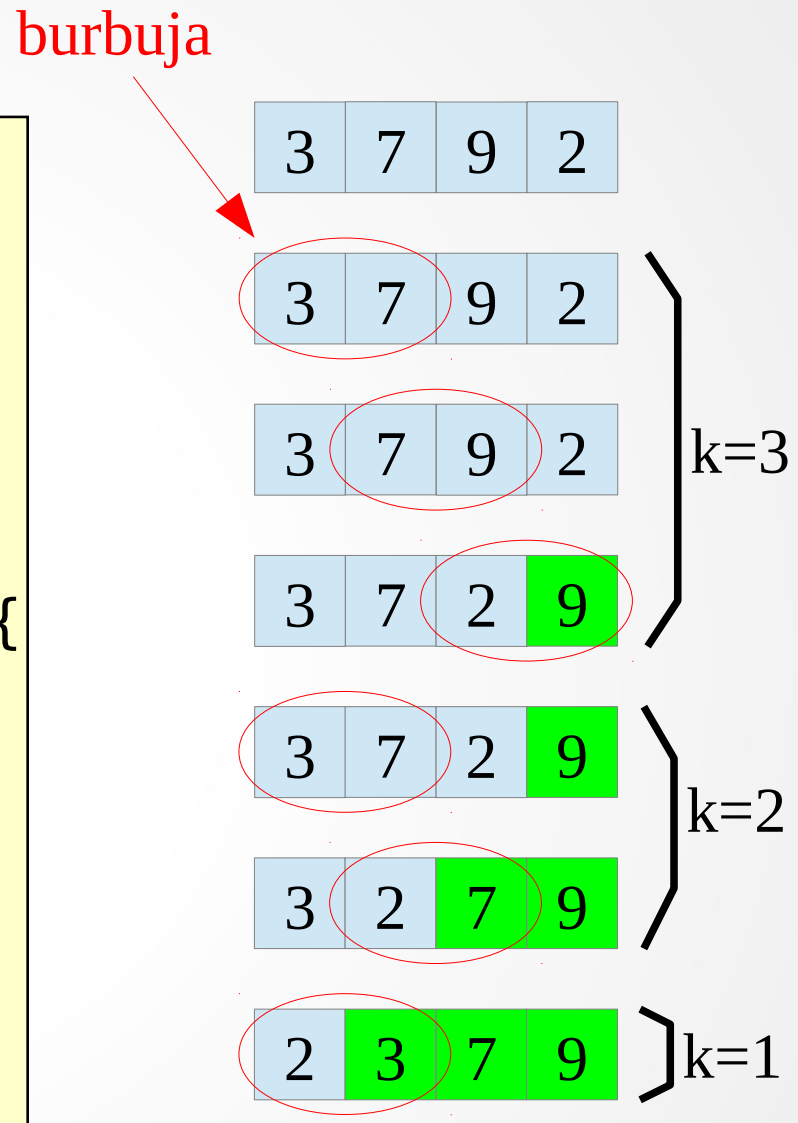
```
int[] vector = ....;
int numElementos = ....;

int pos = ....; // posición del elemento a retirar

for (int k = pos + 1; k < numElementos; k++) {
    vector[k-1] = vector[k];
}
numElementos--;
```

Ordenación por intercambios (burbuja)

- Si el vector no se ha construido ya ordenado, entonces hay que ordenarlo cuando haga falta.
- El método de ordenación más sencillo de programar es el de la **burbuja** (*bubble sort*).
  - Consiste en ir comparando cada pareja de elementos consecutivos, e intercambiarlos si no están en orden.
  - Se realizan pasadas sucesivas sobre el vector hasta que ya esté todo ordenado.
  - En cada pasada el mayor elemento que no esté en su sitio avanza hasta colocarse en su posición.

Ordenación por intercambios (burbuja)

```
int[] vector = ....;
int numElementos = ....;

int final = numElementos - 1;
int aux;
for (int k = final; k > 0; k--) {
  for (int j = 0; j < k; j++) {
    if (vector[j] < vector[j+1]) {
      aux = vector[j];
      vector[j] = vector[j+1];
      vector[j+1] = aux;
    }
  }
}
```



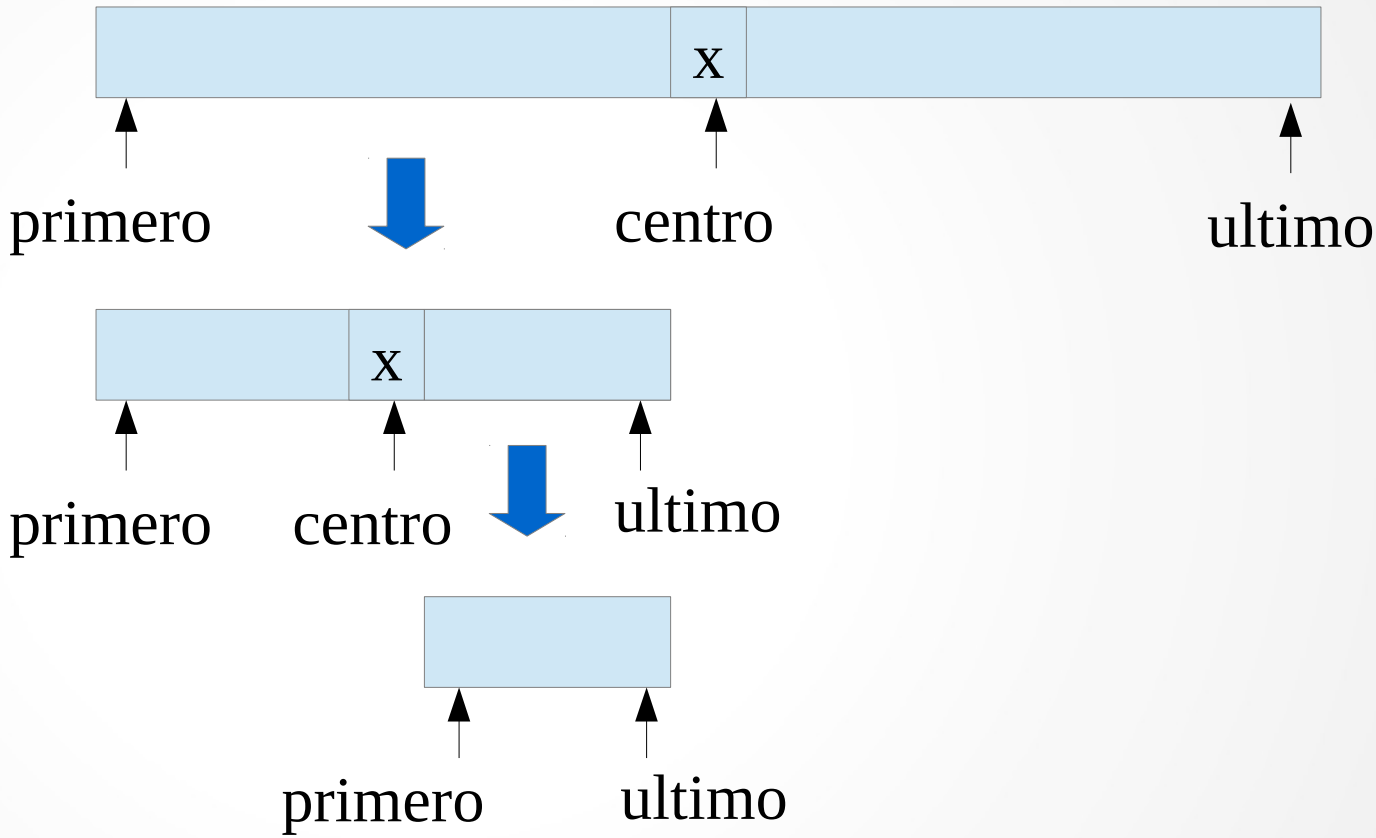
Búsqueda binaria (I)

- La búsqueda simple es poco eficiente. El tiempo de búsqueda es proporcional al número de elementos.
- La búsqueda puede ser más rápida si los elementos están ordenados. Entonces se puede aplicar la búsqueda binaria (búsqueda por dicotomía).
  - La idea es mirar el elemento en posición central. Si no es el buscado se puede **reducir el área de búsqueda** a la mitad anterior o a la mitad posterior, dependiendo de si el elemento central es mayor o menor que el buscado.
  - Este paso de búsqueda **se repite** hasta que se encuentre el elemento o la zona de búsqueda se reduzca a un sólo elemento o ninguno.

Búsqueda binaria (II)

x > buscado

x < buscado



Búsqueda binaria (III)

```
int[] vector = ....;
int buscado = ....;
boolean encontrado;

int primero = 0;
int ultimo = vector.length - 1;
while (primero < ultimo) { // hay más de un elemento posible
    int centro = (primero + ultimo) / 2;
    if (vector[centro] > buscado) {
        ultimo = centro – 1;
    } else if (vector[centro] < buscado) {
        primero = centro + 1;
    } else { // vector[centro] == buscado
        primero = centro;
        ultimo = centro;
    }
}
encontrado = (primero==ultimo) && (vector[primero]==buscado);
```

El operador **<** representa cualquier método de comparación de orden entre los objetos del vector

Ejercicio I

- Añadir al paquete zapatería del proyecto Zapatería la clase Pedido. Un pedido está formado por un array de pedidos simples, un número máximo de pedidos y un contador de los pedidos que se han incorporado al array de pedidos.
- Implementar los atributos necesarios, el constructor y un método incorporar pedidos simples al pedido.
- Implementar un método pedidoProveedor que reciba un proveedor y devuelva el subconjunto del pedido con los pedidos simples de ese proveedor.

Ejercicio II

- Implementar un método diaMayorCaja que devuelve la fecha en la que se encargó el mayor saldo diario de pedidos simples.
- Corregir el programa de prueba hasta que compile y obtenga los resultados correctos esperados.